COSC 120 **Exam 3 Key** Spring 2012

Part 1: Characters and Strings (20 Points)

The shell of a program is below. Write all functions and a main to match the following specifications.

- The `MakeLowerCase` function should convert str to all lowercase.
- The `MakeUpperCase` function should convert str to all uppercase.
- The `RemoveSpaces` function should remove all spaces from str.
- The `LetterCount` function should count and return the number of alphabetic character in the string str.
- The `main` function should declare 3 strings str1, str2, and str3. Strings str1 and str2 should be able to hold an input of up to 50 characters and str3 up to 100. count and return the number of alphabetic character in the string str. The program should read the the first line of input into str1, the second into str2, convert str1 to all lowercase and str2 to all uppercase. It should make str3 the concatenation of str1 and str2. Print out str3 then remove all of the spaces in str3 and print it out again and finally count the number of alphabetic characters in str3 and print that out.

```
#include <iostream>
#include <cstring>

using namespace std;

void MakeLowerCase(char str[])
{
    int i = 0;
    while (str[i] != '\0')
    {
        str[i] = tolower(str[i]);
        i++;
    }
}

void MakeUpperCase(char str[])
{
    int i = 0;
    while (str[i] != '\0')
    {
        str[i] = toupper(str[i]);
        i++;
    }
}

void RemoveSpaces(char str[])
{
    int i = 0;
    int j = 0;
    char str2[strlen(str)+1];

    while (str[i] != '\0')
    {
        if (str[i] != ' ')
        {
            str2[j] = str[i];
            j++;
        }
        i++;
    }
    str2[j] = '\0';
    strcpy(str, str2);
}

int LetterCount(char str[])
{
    int count = 0;
    int i = 0;
    while (str[i] != '\0')
    {
        if (isalpha(str[i]))
            count++;
        i++;
    }
    return count;
}

int main()
{
    char str1[51];
    char str2[51];
    char str3[101];

    cin.getline(str1, 51);
    MakeLowerCase(str1);

    cin.getline(str2, 51);
    MakeUpperCase(str2);

    strcpy(str3, str1);
    strcat(str3, str2);
    cout << endl << str3 << endl;
    RemoveSpaces(str3);

    cout << str3 << endl;
    cout << LetterCount(str3) << endl;

    return 0;
}
```

Input and Output:

```
This IS Test numBER 1234
I Hope I do not need 1235

this is test number 1234I HOPE I DO NOT NEED 1235
thisistestnumber1234IHOPEIDONOTNEED1235
31
```

Part 2: Structures (20 Points)

Create four structs named Point, Line, Rectangle, and Circle. Point should hold two doubles x and y, Line should hold two Points P1 and P2, Circle should hold a Point called center and a double called radius and Rectangle should hold two Points UL and LR (which stand for upper left corner and lower right corner). Then create the following functions,

- CircleArea that takes as input a Circle and outputs the area of the circle. Recall that the area of a circle is πr^2.
- CircleCircumference that takes as input a Circle and outputs the circumference of the circle. Recall that the circumference of a circle is $2\pi r$.
- RectangleArea that takes as input a Rectangle and outputs the area of the rectangle.
- RectanglePerimeter that takes as input a Rectangle and outputs the perimeter of the rectangle.
- LineLength that takes as input a Line and outputs the length of the line. Recall that the length of a line (segment) is the difference in the x values squared plus the difference in the y values squared and then take the square root of the result.

```
const int PI = 3.14159;

struct Point
{
    double x;
    double y;
};

struct Line
{
    Point P1;
    Point P2;
};

struct Rectangle
{
    Point UL;
    Point LR;
};

struct Circle
{
    Point center;
    double radius;
};

double CircleArea(Circle c)
{
    return PI*c.radius*c.radius;
}
```

```
double CircleCircumference(Circle c)
{
    return 2*PI*c.radius;
}

double RectangleArea(Rectangle r)
{
    return fabs(r.LR.x - r.UL.x)
          *fabs(r.LR.y - r.UL.y);
}

double RectanglePerimeter(Rectangle r)
{
    return 2*fabs(r.LR.x - r.UL.x)
         + 2*fabs(r.LR.y - r.UL.y);
}

double LineLength(Line l)
{
    return sqrt((l.P1.x - l.P2.x)
               *(l.P1.x - l.P2.x)
               +(l.P1.y - l.P2.y)
               *(l.P1.y - l.P2.y));
}
```

Part 3: Classes and Objects (25 Points)

Create two classes named Point and Rectangle. Point should hold two doubles x and y, and Rectangle should hold two Points UL and LR (which stand for upper left corner and lower right corner).

- The Point class should have a default constructor, constructor that brings in x and y, and accessor functions.
- The Rectangle class should have a default constructor, constructor that brings in two points, accessor functions and the functions Area that outputs the area of the rectangle and Perimeter that outputs the perimeter of the rectangle.

All data members must be private. Write both the declaration and implementation for the classes.

```
class Point
{
public:
    Point();
    Point(double px, double py);

    double getX();
    double getY();

    void setX(double X);
    void setY(double Y);

private:
    double x;
    double y;
};

Point::Point()
{
    x = 0;
    y = 0;
}

Point::Point(double px, double py)
{
    x = px;
    y = py;
}
```

```
double Point::getX()
{
    return x;
}

double Point::getY()
{
    return y;
}

void Point::setX(double X)
{
    x = X;
}

void Point::setY(double Y)
{
    y = Y;
}
```

```
class Rectangle
{
public:
    Rectangle();
    Rectangle(Point ul, Point lr);

    Point getUL();
    Point getLR();

    void setUL(Point p);
    void setLR(Point p);

    double Area();
    double Perimeter();

private:
    Point UL;
    Point LR;
};

Rectangle::Rectangle()
{
    UL.setX(0);
    UL.setY(0);
    LR.setX(0);
    LR.setY(0);
}

Rectangle::Rectangle(Point ul,
                          Point lr)
{
    UL = ul;
    LR = lr;
}
```

```
Point Rectangle::getUL()
{
    return UL;
}

Point Rectangle::getLR()
{
    return LR;
}

void Rectangle::setUL(Point p)
{
    UL = p;
}

void Rectangle::setLR(Point p)
{
    LR = p;
}

double Rectangle::Area()
{
    return fabs(LR.getX() - UL.getX())
          *fabs(LR.getY() - UL.getY());
}

double Rectangle::Perimeter()
{
    return 2*fabs(LR.getX()-UL.getX())
         + 2*fabs(LR.getY()-UL.getY());
}
```

Part 4: Pointers (20 Points)

1. Give the output of the following code. If the output is an address simply say that it is the address of and then give the number that is in that address location.

```
#include <iostream>

using namespace std;

void fct1(const double *list, int size)
{
    for (int i = 0; i < size; i++)
        cout << *list++ << "  ";
}

void fct2(double *list, int spot, double num)
{
    *(list+spot) = num;
}

int main()
{
    double *arr;
    arr = new double[50];

    if (arr == NULL)
    {
        cout << "Memory Allocation Error" << endl;
        return 1;
    }

    fct2(arr, 2, 12.5);
    fct2(arr, 1, 1);
    fct2(arr, 5, 7.4);
    fct2(arr, 0, -9.3);
    fct2(arr, 3, 10.1);
    fct2(arr, 4, 15.75);

    fct1(arr, 6);
    cout << endl;

    cout << arr[5] << endl;
    cout << *(arr+3) << endl;
    cout << (arr+4) << endl;
    cout << &arr[2] << endl;
    cout << *arr << endl;
    cout << *(&arr[5]-1) << endl;

    delete [] arr;

    return 0;
}
```

```
-9.3  1  12.5  10.1  15.75  7.4
7.4
10.1
0x381020  --- position of 15.75
0x381010  --- position of 12.5
-9.3
15.75
```

2. Give the output of the following code.

```
#include <iostream>
#include <cmath>

using namespace std;

double square(double a)
{
    return a*a;
}

double cube(double a)
{
    return a*square(a);
}

double mathfct(double (*fct)(double), double num)
{
    return fct(num);
}

int main()
{
    cout << mathfct(square, 3) << endl;
    cout << mathfct(cube, 2) << endl;
    cout << mathfct(sqrt, 100) << endl;
    cout << mathfct(sin, 3.14159) << endl;
    cout << mathfct(cos, 3.14159) << endl;

    return 0;
}
```

```
9
8
10
0
-1
```

Part 5: Operator Overloading (25 Points)

Update the Point class from the previous exercise so that it adds the following. Write just the implementation for the class.

- Overload + so that it adds the points componentwise. (x1, y1) + (x2, y2) = (x1+x2, y1+y2)
- Overload – so that it subtracts the points componentwise. (x1, y1)–(x2, y2) = (x1–x2, y1–y2)
- Overload the assignment operator =.
- Overload the = = operator.
- Overload * operator to do the dot product. (x1, y1)*(x2, y2) = x1*x2 + y1*y2, Note that the output is a double.
- Overload the cout operator <<.

```
Point Point::operator+ (const Point &rhs)
{
    Point retpt(x+rhs.x, y+rhs.y);
    return retpt;
}

double Point::operator* (const Point &rhs)
{
    return x*rhs.x + y*rhs.y;
}

Point Point::operator- (const Point &rhs)
{
    Point retpt(x-rhs.x, y-rhs.y);
    return retpt;
}

Point Point::operator= (const Point &rhs)
{
    x = rhs.x;
    y = rhs.y;

    return *this;
}

bool Point::operator== (const Point &rhs)
{
    return (x == rhs.x) && (y == rhs.y);
}

std::ostream& operator<< (std::ostream& output, const Point &rhs)
{
    output << "(" << rhs.x << ", " << rhs.y << ")";
    return output;
}
```